Final Report*

for

Grant No. N00014-87-K-0796

Design and Analysis of Scheduling Policies for Real-Time Computer Systems

J.F. Kurose, C. M. Krishna, D. Towsley
University of Massachusetts
Amherst, MA 01003

kurose@cs.umass.edu, krishna@ecs.umass.edu, towsley@cs.umass.edu

1 Introduction

Our research funded under ONR contract N00014-87-K-0796 can be broadly divided into five areas:

1. design and analysis of deadline based scheduling policies,
2. design and evaluation of high performance and fault tolerant disk architectures for real-time systems,
3. design and evaluation of scheduling policies for real-time tasks with incremental-value-with-increased-execution-time characteristics,
4. reliability and testing of real-time systems,
5. scheduling for real-time parallel processing systems.

These topics will be the subject of the remainder of the technical section. Additional details of our work can be found in the cited technical papers and reports.

2 Deadline Based Scheduling

In soft real-time computer and communication systems, temporal constraints are placed on the behavior of the jobs (e.g., processes or messages) within these systems. Typically, these constraints require that these jobs initiate or complete some task (e.g., a process' computation or a message's transmission) within some *deadline*. In such soft real-time systems, the performance metric of interest is no longer one of the traditional measures, such as average delay or throughput, but rather, the fraction of jobs which are not able to meet their specified time constraints (i.e., the fraction of jobs that are *lost*.)

One of our primary goals during this last year was to understand the behavior of two scheduling policies in such a system with deadlines: the minimum laxity policy (ML) and earliest deadline policy (ED). In the ML policy, deadlines are to the beginning of service; in the ED policy, deadlines are to the end of service. Both policies schedule that job whose deadline is closest to expiring. In the following subsections, we overview the results that we have obtained regarding the optimality (in terms of minimizing loss) of these policies and then describe two approaches for modeling their performance.

2.1 Optimality of the ML and ED Policies

We have been able to establish the following optimality property of the ML and ED policies on a multiprocessor executing a stream of jobs with arbitrary arrival times and deadlines. Assuming that job service times are exponentially distributed, we established that over the entire class of non-idling scheduling policies, both the ML and ED policies maximize the fraction of jobs that meet their time constraints.

These results hold both for the case that an unlimited number of jobs can reside in the system and for the case that a maximum of B jobs can reside in the system at any one time. In such systems, the "scheduling" policy must also determine which job should be removed from the system whenever it is full and a new job arrives. We have shown that the policy that removes the job closest to its deadline, coupled with the ML or ED policy for scheduling jobs, maximizes the fraction of jobs that make their deadlines under the same assumptions as above. Details of these results may be found in [26]. We note that these results are particularly powerful as they establish the optimality of two specific scheduling disciplines over a large class of possible real-time scheduling disciplines.

We have also obtained similar results for the case that deadlines are not precisely known but a "stochastic ordering" exists among the deadlines of all jobs in the system. Details of these results may be found in [26].

We have also treated systems in which customers are not removed when they miss a deadline. See [7] for details.

2.2 Bounds on the Performance of the ML and ED Policies

In addition to studying the optimality properties of the ML and ED policies, we also considered two approaches towards evaluating their performance. In our first approach, we developed a Markovian model that describes the behavior of ML on a multiprocessor under the assumptions of Poisson arrivals and exponentially distributed service times and deadlines. A similar model was developed for the ED policy for a single processor system under identical assumptions. Unfortunately, an *exact* analysis of this model is computationally intractable (from a practical standpoint). However, we were able to develop tractable models which produce upper and lower bounds on the fraction of jobs lost. These bounds can be made arbitrarily tight at the cost of additional computation. The results of this analysis can be found in [12].

The Markov model underlying these bounds is based on a *new binary simulation of the ML and ED policies*. This simulation can be found in [11] and can be used to develop models for ML and ED scheduling for the cases in which the service times are generally distributed, or the arrival times are generally distributed.

2.3 Exact Analysis of the ML Policy

The performance analysis described in the previous section was approximate and required exponential assumptions for the interarrival times, service time, and deadline distributions. We have also developed a computational algorithm for *exactly* computing customer loss under less restrictive assumptions, provided the system can be modeled as a *discrete time* queueing system. Such a model would be appropriate in communication networks and computer systems in which event timings occur in discrete units of time (e.g., any time-division-multiplexed communication link/network or computer system in which job execution times and interarrival times are multiples of some minimum time quantum).

Specifically, we considered a discrete-time queueing system in which the *deadline* associated with each customer (the amount of time from a customer's arrival until the time at which it must

begin service) is bounded by some maximum possible value, M time units.. Customers in the queue are scheduled according to the ML policy, and a customer whose deadline expires is considered lost and is removed from the queue without receiving service. We have been able to exactly analyze the case of geometrically distributed service times and a bulk arrival process in which the number of customers arriving in a slot with a deadline of i slots is also geometrically distributed (for each $i, 1 \leq i \leq M$); we have also demonstrated how this model can be extended to include generally distributed service times and laxities as well. The main result of this work has thus been the development of a numerical algorithm which exactly computes customer loss for this queueing system with a time complexity of $O(M^4)$. Details of this work can be found in [17].

2.4 Approximate ML and ED policies

One potential drawback of ML scheduling is that the identity of the job with the closest deadline (minimum laxity) must be determined at each scheduling point – a potentially expensive run-time cost, especially when the number of queued jobs is large. For example, if jobs are maintained in a list structure, finding the minimum laxity job in a non-laxity-ordered list or maintaining a sorted list according to laxity are both $O(n)$, when there are n jobs queued; if a dictionary-like structure is used to queue jobs, the time to maintain the data structure is $O(ln(n))$.

We developed a policy ML(n) [12] that approximates the behavior of ML in the following way. This policy divides the overall queue into two queues, $Q1$ and $Q2$, where $Q1$ can hold at most n jobs. If the total number of jobs waiting for service is less than or equal to n, they are all held in $Q1$. Jobs in $Q1$ are scheduled according to ML. However, if the number of jobs exceeds n, then an arriving job is unconditionally placed into $Q2$. At a service completion instant, the job with minimum laxity among all jobs queued in $Q1$ is scheduled for service. When the scheduler moves a job from $Q1$ to the server, it also moves a job from $Q2$ to $Q1$, selecting the job to enter $Q1$ on an FCFS basis. In summary then, $Q1$ is an ML queue of size n and $Q2$ is an FIFO queue of unbounded size and $Q2$ feeds $Q1$. Note that ML(1) is same as FCFS and ML(∞) is equivalent to exact ML.

In [20] we compared this policy with a variant, first presented in [29] which reverses the positions of $Q1$ and $Q2$, i.e., the ML portion of the queue, $Q1$, feeds the FIFO portion of the queue, $Q2$. We showed that these two seemingly dissimilar policies always make *the same scheduling decisions at the same time.*

In [10], we consider four variants of the ML(n) policy that approximate the behavior of ML scheduling and continue to enjoy the advantage of having a run-time cost which is independent of the number of queued customers. Our simulation results show that the best of the four policies provides 20–25% improvement over ML(n) and performs within 5% of the exact ML policy over a wide range of traffic loads and laxity distributions. This policy differs from ML(n) in the following manner. At arrival, if there are n or more jobs in the system, the laxity of the new arrival is compared to the laxity of the job with the *maximum laxity* among the n jobs in $Q1$. If the laxity of the new arrival is greater, the new arrival is placed at the *end* of $Q2$; otherwise, the new arrival is placed in the position of the job with maximum laxity among the jobs in $Q1$, and the job with maximum laxity in $Q1$ is placed at the *end* of $Q2$.

Details of the other policies and their performance evaluation can be found in [10].

3 High Performance and Fault Tolerant Real-Time I/O Systems

Throughout our contract, we have concerned ourselves with the development of real-time fault tolerant high performance I/O systems.

3.1 Scheduling policies for real-time disks

We developed and evaluated the performance of two new disk scheduling algorithms for real-time systems. These algorithms, called SSEDO(for *Shortest Seek and Earliest Deadline by Ordering*) and SSEDV(for *Shortest Seek and Earliest Deadline by Value*), combine *deadline* information and *disk service time* information in different ways. The basic idea behind these new algorithms is to give the disk I/O request with the earliest deadline a high priority; but if a request with a larger deadline is "very" close to the current disk arm position, then it may be assigned the highest priority. The performance of SSEDO and SSEDV algorithms is compared with three other proposed real-time disk scheduling algorithms, ED, P-SCAN, and FD-SCAN, as well as four conventional algorithms, SSTF, SCAN, C-SCAN, and FCFS. An important aspect of the performance study is that the evaluation is not done in isolation with respect to the disk, but as part of an integrated collection of protocols necessary to support a real-time transaction system. The transaction system model was validated on an actual real-time transaction system testbed, called RT-CARAT. The performance measures of interest are the transaction loss probability and the average response time for the committed transactions under different I/O scheduling algorithms. The results show that SSEDV outperforms SSEDO; that both of these new algorithms can improve performance of up to 38% over previously-known real-time disk scheduling algorithms; and that all of these real-time scheduling algorithms are significantly better than non-real-time algorithms in the sense of minimizing the transaction loss ratio. Details of this study can be found in [3].

Although the performance evaluation was done in the context of a transaction processing system, the policies can be used in any real-time setting and the relative rankings of all of the policies studied should not be effected.

3.2 Fault tolerant disk systems

We developed and evaluated the performance of a number of scheduling algorithms for a pair of disks where copies of each data item are maintained on each disk. We considered two classes of policies, *i) centralized queue policies* (CQP's) and *ii) distributed queue policies* (DQP's). A CQP maintains a central queue of all requests that require servicing by the mirrored disks. In addition, a second, auxiliary queue may form from time to time at the disk that lags behind. When a disk becomes available, a request is scheduled to it from the central queue according to some policy. If the request is a read, then it is served by this disk. If it is an update, then in addition to being served by the available disk, it is also entered into the auxiliary queue associated with the second disk. Last, requests are scheduled from the auxiliary disk according to some policy.

A DQP maintains a separate queue at each disk. A read request is assigned to one of these disks according to some routing rule, whereas an update generates write requests at both disks. Requests are scheduled at each queue according to some scheduling policy. Last, whenever a read arrives to find both disks idle, it is routed to the one that will provide the shortest seek time.

Our studies have focussed on the choice of scheduling policies at the queues and the choice of routing policy in the class of DQP's. We have concluded that the best performance (i.e., smallest fraction of jobs missing their deadlines) is obtained with a DQP which uses a *join the shortest queue* routing policy for reads and the SSEDO (shortest seek and earliest deadline by ordering) policy at each of the queues. Details of this work can be found in [4, 24]; see also [3] for details on SSEDO.

3.3 Disk arrays

Our work in this area has focussed on developing and evaluating the performance of different data layout schemes and scheduling policies for two proposed disk array architectures, the *mirrored array* and *rotated parity array*.

In both cases, the system consists of N disks. The mirrored array ensures that there are two copies of each file, whereas the rotated parity array provides a parity block for fault tolerance for every $N-1$ data blocks.

In the case of the mirrored array, we proposed several data allocation schemes and scheduling policies We compared their performance in the case that all disks are operational (normal mode) under two workloads: *i)* applications in which I/O requests are for small amounts of data (e.g., transaction processing, workstation), and *ii)* applications in which I/O requests are for large amounts of data (e.g., supercomputing, image processing). The main results is that in the normal mode of operation, a newly-proposed *group-rotate declustering* allocation, coupled with a policy that assigns read requests to the disk containing the data with the shortest queue, provides the lowest mean response time of all of the combinations that we considered. This is true for both types of applications described above.

In the case of the rotated parity array, we compared the performance of the traditional RAID 5 layout where files are interleaved across the disks, with the parity striping layout, where each file is stored on a single disk. We also studied two synchronized I/O scheduling policies suitable for both layouts, which take care of this problem. These two policies provide practical solutions to the problem of providing write synchronization in rotated parity arrays. We provided accurate mathematical models for estimating the mean I/O response times and the maximum throughput of both layouts, coupled with these two synchronized scheduling policies. Using these models, we compared the performance of the two layouts with each other. The results show that the performance of RAID 5 (with relative small striping unit of 4K bytes) is sensitive to the increase of mean request size but not to the skew in the access pattern. On the other hand, the parity striping layout is sensitive to skew in the access pattern. Therefore, depending on applications, RAID 5 outperforms parity striping in some cases, but is outperformed by parity striping in other cases. We identify workloads for which each layout provides the best performance. This allows the designer the ability to choose between them for their applications.

Last, we compared the two arrays with each other and observed that the mirrored array archi-

tecture significantly outperforms the rotated parity array architecture when applications generate I/O requests for small amounts of data. This is true for the case that both architectures have the same number of disks as well as when they have the same storage capacity. In the case of applications that generate I/O requests for large amounts of data, the results are not as clear. RAID 5 performs better when most requests are very large, most requests are writes, and most writes perform *full stripe writes*.

Similar results have been obtained in the case of soft real-time workloads. Details of this and the above work can be found in [5, 6, 25].

4 Scheduling Real-Time Tasks with Incremental Reward Characteristics

Many real-time systems can be modeled by a single server system in which jobs arrive (according to a certain distribution) and remain in the system for a certain amount of time before departing. The longer you serve a job (while it is present in the system) the more profit you make on the job. In typical problems (such as in Artificial Intelligence) the profit curves are concave [2].

In this work, we considered the problem of scheduling jobs whose arrivals are described by an arbitrary stochastic process. We assume that job i has a lifetime of τ_i and a profit curve $f_i(x_i)$ which is an increasing concave function of x_i the received service.

The objective is to schedule the jobs to maximize the profit per unit time. We refer to the lifetime τ_i of job i as its *initial laxity*. At any time t, let a job be present in the system with deadline d ($d > t$). The remaining time until the job leaves the system is referred to as the *laxity* (as opposed to *initial laxity*) of the job. Thus, at time t the laxity of a job with deadline d is $l = d - t$.

We have formulated and solved a static optimization problem where we assume the presence at time $t = 0$ of N jobs with distinct deadlines and profit functions for the case of no future arrivals. In the case of arbitrary concave profit functions, the optimal schedule can be obtained with an amount of computation that is $O(N^2)$. If we assume that the profit curve is of the form $f_i(x) = 1 - e^{-\delta_i x}$, then the complexity is reduced to $O(N \log N)$. We have developed an algorithm that accounts for arrivals, by executing this static policy at each arrival epoch and using the associated schedule until the following arrival. As this heuristic does not produce a unique schedule to be used during this period, we have considered the following heuristics.

1. **Shortest Service** - In any given interval, service the job with the least service so far.
2. **Earliest Deadline (ED)** - In any given interval, service the job with the closest deadline.
3. **Random Selection** - In any given interval, service a job randomly.

We have compared these different heuristics to each other, to simple first-come-first-serve and last-come-first-serve policies, and to simple upper bounds in the case that two different classes of customers are being served. Here, customers from different classes differ in their profit function and

laxity distributions. We find that the simple ED policy, in combination with the static optimization algorithm, performs the best and achieves a performance close to the unachievable upper bound in most cases.

These algorithms are likely to be of importance in many real-time applications which involve successive approximations or search procedures [2, 8, 9, 19].

5 Reliability and Testing

5.1 Reliability Modeling of Real-Time Systems with Transient and Correlated Failures

Real-time systems can fail not just because of spontaneously-occurring failures, but also because of events, such as a burst of electromagnetic or elementary-particle radiation. Such environmental upsets can cause both transient and permanent failure. Since the entire system is bathed in the same environment, such failures can be correlated. However, despite the occurrence of correlated failures, contemporary reliability models have largely ignored them, assuming instead that failures are independent.

Our work on developing the mathematical underpinnings of a model which accounts for both correlated and independently-occurring failure has been described in an earlier report. Over the past year, we have completed work on the first version of a software package which implements this earlier research.

The package, written in C, considers only processor failures: the failure of the software, the I/O components, etc., is not considered. Our future work will extend it to include these factors.

The package accepts as input the number of processors, the inter-checkpointing interval, the spontaneous permanent and transient failure rates of processors, the critical workload schedule of each processor, and the characteristics of the operating environment. The latter is represented by the transition rates of the environment between a variety of states. Each of the environmental states represents some degree of stress imposed by the environment on the computer system, and thus represents a given increase in the permanent or transient failure rates. The package allows the system to be heterogeneous, i.e., the failure rates and susceptibility to the operating environment can vary from processor to processor. The program output is the probability that the critical workload is executed on time throughout a specified interval of operation.

The features of this package which distinguish it from others is that *(i)* correlated failures and *(ii)* the periodic critical workload schedule are taken into account. It can be used, for example, to evaluate the performance of different task schedules (including staggered schedules) and the effects of shielding against the effects of the environment.

5.2 Scheduling Tasks on Real-Time Systems Subject to Correlated Failure

In this part of the work, we studied real-time systems operating in hostile environments [15]. We have obtained a heuristic to carry out scheduling of tasks on real-time systems that are subject to correlated transient failure. The tasks have cost functions associated with them. That is, $c_i(t)$ is

the cost of having a response time t for an iteration of task i. These functions relate the cost to the controlled process of the response time of the various tasks. The objective is to minimize the total cost and maximize the available time redundancy (required to defeat correlated transient failure), subject to the requirement that all deadlines be met. Since this problem is clearly NP-hard, we have focussed on developing a heuristic. The heuristic is in two parts.

At each decision point (when a new task becomes available or a currently-executing task completes), the system has to determine which of the available tasks must be run. To obtain a minimum-cost schedule would require us to generate a search tree of as many levels as there are tasks, and exhaustively evaluate it. This, however, is very expensive, and so only a subset of the entire set of available tasks must be considered. To determine which tasks are to be in this set (called the *lookahead set*), we use the following procedure for the first part of the heuristic.

From the set of tasks that are available for dispatch at decision time t (called the *current set*, $C(t)$), choose the task, i, which is most expensive to execute last (i.e., after all the other tasks in $C(t)$). Find the slope of the cost function of task i both at time t as well as at the time when all the other tasks in $C(t)$ finish. Call these slopes s_{i1} and s_{i2}, respectively. Include in the lookahead set task i, as well as those tasks in $C(t)$ whose cost functions at time t have slopes greater than $\min\{s_{i1}, s_{i2}\}$. This lookahead set is then searched exhaustively to determine which task should be dispatched.

Note that the above actions only check the slope of the cost function for task i at two time instants. Since cost functions can be highly nonlinear, this can lead to anomalies. The second part of the heuristic, called the compaction step, attempts to correct these anomalies. Compaction starts with the schedule for the entire task set, obtained as described above. The last task is deleted from the task set, and a new schedule is obtained for the rest of the task set. This last task is then inserted in the first unused miniframes during which the task is available. Note that the generation of the new schedule is done recursively, with the last task in each schedule being deleted from the entire task set at each point until no tasks remain, and then the schedule is gradually built up by reinserting the deleted tasks as specified above.

We have run extensive simulations to compare the quality of the schedules generated by this algorithm against those of the optimal algorithm (using exhaustive enumeration). In the vast majority of cases, our algorithm produces schedules whose costs are within 5% of the optimal.

So far, we have said nothing about maximizing the time redundancy, i.e., the amount of slack in the schedule after all the critical tasks have been inserted. To do this, we proceed as follows. Suppose there are $2m + 1$ copies of each task that have to be run. Then, we first schedule only $m + 1$ copies of each task. Following this, we add to the schedule an $(m + 1 + i)'$th copy of each task, and place it in the schedule appropriately, for $i = 1, 2, ..., m$. In every case, the placement of the $(m + k)'$th copy of each critical task is done prior to the introduction into the schedule of the $(m + \ell)'$th copy for $\ell > k$, and the positioning of the $(m + k)'$th copy of any task is unaffected by the positioning of the $(m + \ell)'$th copy of any other task except insofar as may be required to meet the deadlines of all the tasks.

This results in the tasks being staggered in such a way that the $(m + k)'$th copy of each task is completed before the $(m + \ell)'$th copy of the same task (for $\ell > k \geq 1$). As a result, whenever a

total of $m + 1$ identical outputs are produced by the first $m + 1$ correctly functioning copies of a task, the rest of the copies can be dispensed with, thus generating additional slack in the schedule.

5.3 Impact of Workload on the Reliability of Real-Time Systems

Most real-time systems employ N-modular – most commonly triple-modular – redundancy for fault-tolerance. When a processor in a triad fails permanently, a spare processor (if available) must be switched in to take its place. If the failure is transient, the affected processor will be brought back into the triad after it recovers.

In either case, it is necessary to make the memory of all three members of the triad consistent. This can be done by copying into the recovering or substitute processor the writeable memory of the two processors that are still functional. The time required for this can depend on the workload, and the rate at which this workload writes into its memory.

Until the processor has recovered, is resynchronized with its colleagues in the triad, and resumes normal operation, the triad is effectively a duplex and will suffer fatal failure if one more of its processors fails.

In this work, we have modeled the impact of workload on the recovery time, and therefore on the reliability, of processor triads. We have shown that there is a knee above which the allocation of more tasks to processors increases the fatal failure rate dramatically. Our current work deals with the implications of this fact on the allocation of tasks to real-time systems. Details can be found in [16].

5.4 *Distributed Recovery Algorithms* for Distributed Systems

One component in our initial proposal was the development and performance evaluation of distributed algorithms for recovering from faults in a distributed real-time computer system. Briefly, the algorithm that we proposed requires that a node, henceforth referred to as the primary node, transmit one or more copies of a job at the time of its arrival to other nodes, referred to as *secondary nodes*, in the system. The secondary nodes are responsible for monitoring the primary node for failure. If a failure occurs before the job completes, the secondary nodes select one of them to be the new primary node responsible for completion of the job.

There are many interesting variations of this basic policy, and thus our first task was to develop a simple analytical model which can be used to study the performance of these variations. We have chosen an approach whereby we decompose the system of N nodes into N models, one for each node in the system. The interactions between these nodes are captured by the values of the input parameters of each of these models. As these parameter values are unknown, this yields a fixed point problem, i.e., a set of nonlinear equations with these parameters as unknowns. We have developed a detailed model for a single node which can be used in such an approximate evaluation; our model accounts for the effects of node failures and the communication costs of transferring job copies from the primary node to the secondary nodes. The details of this model and its analysis can be found in [27].

5.5 Scheduling Tests of Software for Real-Time Systems

The reliability of real-time systems depends greatly on the reliability of the applications and systems software that is run on it. Two approaches to reliable software have been proposed in the literature. The first approach, called the *recovery block approach*, deals with using a primary version and a secondary (or backup) version. There is an acceptance test whose function is to determine whether or not the output is likely to be correct. The acceptance test is not perfect: it is assumed to have a probability of $c < 1$ of detecting an erroneous output. c is called the *coverage*. If the primary is judged to have produced an erroneous result, the secondary is invoked.

The second approach, known as N-version programming, is a software analogue of N-modular redundancy. N versions of the software are independently produced and run in parallel. The results of the software are voted as in N-modular redundancy.

In this work, we provide a simple reliability model for N-version programming and the recovery block scheme which can provide guidance for the quasi-optimal allocation of software debug time among the different versions [28].

We assume that the software error generation rate is a Weibull function of the debug time t, i.e.,

$$\lambda(t) = \lambda_0 \, e^{-(\mu t)^\alpha}$$

where μ and α are constants that characterize the software being debugged, and λ_0 is the initial failure rate of the software. The Weibull distribution was chosen because of its generality. Note particularly that the popular exponential distribution is a special case of the Weibull.

We have obtained expressions for the Mean Time to Failure (MTTF) of both schemes as a function of the debug time of the various modules. In particular, our results show that when the coverage of the acceptance test in the recovery-block approach is imperfect (i.e., $c < 1$), most of the debug time should be spent on the primary. Indeed, the share of the debug time allocated to the primary tends to increase as the coverage decreases. The expression for the MTTF of N-version programming is too complex to maximize analytically: instead we show how to use numerical techniques to allocate the debug time optimally.

5.6 Optimal Scheduling of Signature Analysis Tests

A second effort in the area of reliability has studied methods to optimally schedule tests in real-time systems. Fault-tolerant systems need to undertake regular and mutual testing to flush out latent faults and reconfigure the system in response. The use of ***signature analysis*** as a testing procedure has gained rapidly in popularity over the last few years. Signature analysis consists of applying a sequence of test inputs to the device under test, and compressing the outputs. This compressed output is then compared against a reference. Any discrepancies would indicate a faulty device.

The convention is to apply the entire test sequence to the device and then compare it against the reference. However, if a fault is uncovered early, the rest of the test can be dispensed with, thus saving time. It therefore makes sense to embed additional comparisons against the reference, i.e., to break the sequence of tests down into subsequences. At the end of each subsequence, if

any faults have been uncovered, the testing stops since the device under test is faulty and must be purged; otherwise, the next subsequence (if one is available) is applied.

Our accomplishment has been to obtain an algorithm which breaks the test sequence down into a set of subsequences so that the expected testing time per device is minimized. Our work is thus likely to reduce the testing overhead in operational real-time systems. Over the past year, we have tested this algorithm on the benchmark circuits of Brglex, et al., and showed its practical usefulness. The inputs to our algorithm are: the probability that the device under test is faulty, the coverage function of the test sequence, and the overhead consumed in applying tests to comparing the test outputs against the corresponding response. This work is described in detail in [18]. We plan to extend this work to include board-level diagnosis. This will involve incorporating search algorithms in our work to locate a test input out of the whole sequence that exercises this fault.

5.7 Optimizing Wafer-Probe Testing

The VLSI chips that make up highly-reliable systems must be thoroughly tested during manufacture to ensure that defect levels[1] are suitably low. Unfortunately, exhaustive testing is out of the question and even the best testing procedure is imperfect; that is, failed chips can pass the test and be incorporated into a product.

Thus, it is always of interest to *(i)* optimize the test effort required to achieve a given defect level, and *(ii)* to achieve the lowest possible defect levels given the best available (imperfect) testing procedure. In this work, we have developed a novel approach for improving the effectiveness of wafer-test procedures by obtaining and using yield estimates for individual dies on the wafer before the wafer is diced.

Our approach is based on the observation that defects on a wafer are not uniformly distributed, but have long been known to exhibit clustering. Most of the good dies on a wafer are found adjacent to other good dies, while defective dies tend to be similarly clustered. This suggests that if we know the state of some or all of the neighbors of a given die, we can obtain a better estimate of its yield. We have shown how to use this improved yield estimate to optimize the test applied to the die. We have calculated that in typical cases, we can better manage the test process to obtain, for the same testing time, a halving of defect levels, and that we can identify dies whose defect level is about *20 times* less than for the overall lot.

6 Parallel Systems

As part of our research, we designed and evaluated the performance of scheduling policies for parallel systems executing jobs with real-time constraints. We have focused on two aspects of this problem:

- The analysis of priority scheduling policies for multiprocessors executing parallel applications.
- The determination of optimal scheduling policies for (both real-time and non-real-time) parallel processing systems executing parallel applications.

[1]The defect level is the ratio of bad chips which pass the test to all the chips that pass the test.

6.1 Priority Policies for Multiprocessors

We have developed simple models for a multiprocessor which executes a stream of K classes of jobs, each of which consists of a random number of tasks that can be executed independently of each other; we refer to such jobs as a *fork-join* jobs. Several priority scheduling policies have been analyzed: a) a strict non-preemptive head of the line policy, b) a preemptive policy that allows preemptions at the job level, c) a preemptive policy that allows preemptions at the task level, and d) a policy where the priority is a non-decreasing function of the number of tasks in the queue with preemptions at the job level. Using these models, we have compared the mean job response time for the different classes under the various scheduling policies and under FCFS scheduling. We have also compared the performance of these policies to that of a system in which the processors are partitioned so that classes are allocated only to certain processor groups. Our results have shown that for the system considered, the task preemption policy has a uniformly better class response time and thus is preferable to a system with partitioned processors. Details of this study will be available in a forthcoming report [22].

We are also now completing a related study of the behavior of the first-come-first-serve scheduling policy for fork-join jobs on a multiprocessor. Our results here have included a characterization of the response time distribution under Markovian assumptions, development of computationally efficient upper and lower bounds for the moments of the response time, a proof that FCFS is the policy that minimizes (and last-come-first-serve (LCFS) maximizes) the expected value of a convex function of the response time. This last property has the implication that FCFS minimizes (and LCFS maximizes) any moment of the response time distribution. From a practical standpoint, this means that LCFS maximizes the fraction of jobs that complete within their deadline in a real-time system in which all jobs eventually receive service (whether or not they miss their deadlines) under a general set of assumptions. Details of this study can be found in [21]. We note once again that we believe this is a strong result as it establishes the optimality of LCFS over a broad range of possible scheduling disciplines.

6.2 Optimality Results

We also completed a study of the effects of scheduling disciplines on the performance of parallel systems (both with and without real-time constraints). A job is composed of a set of tasks, with a partial order specifying the precedence constraints between the tasks. We assume that a predefined mapping of the tasks to processors has been given and that the processors execute a stream of jobs, all with the same task graph and task/processor allocation. Our goal is to study the effects of different *local* scheduling policies at each of the processors on the job throughput, number of jobs in the system, and the job response time.

We have been able to establish several important results. First, we have been able to show that the FCFS policy applied at the task level minimizes the number of jobs in the system and maximizes the throughput. Second, we have also established that FCFS minimizes the expected value of any convex function of the response time. In the case that jobs have soft real-time deadlines and are completed regardless of whether they make their deadlines, we have shown that the earliest deadline policy (ED) minimizes the expected value of any increasing convex function of the *lag time*,

where the lag time is defined as the difference between a job's deadline and its actual completion time. The latest deadline policy (LD) has also been shown to maximize this measure. Finally, we have been able to establish that the LCFS scheduling policy maximizes the fraction of jobs that complete by their deadlines among the class of policies that do not use service time or deadline information. Details of this research may be found in [1].

References

[1] F. Baccelli, Z. Liu, D. Towsley, "Optimal Scheduling of Parallel Processing Systems with Real-Time Constraints", to appear in *J. ACM.*

[2] M. Boddy and T. Dean, "Solving Time-Dependent Planning Problems," *Proc. Eleventh Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, August 1989, pp. 979-983.

[3] S. Chen, J.A. Stankovic, J.F. Kurose, D. Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems", to appear in *J. of Real-Time Systems.*

[4] S. Chen, D. Towsley, "Performance of a Mirrored Disk in a Real-Time Transaction System", *Proc. 1991 ACM SIGMETRICS Conference,* May 1991.

[5] S.-Z. Chen and D. Towsley, "Raid 5 vs. parity stripping: their design and evaluation," to appear in *Journal of Parallel and Distributed Computing*, January 1993.

[6] S. Chen, D. Towsley, "A Queueing Analysis of Disk Array Architectures," submitted to *IEEE Trans. on Computers.*

[7] S. Chen, D. Towsley, "Scheduling Customers in a Non-Removal Real-Time System with an Application to Disk Scheduling", submitted to *Journal of Real-Time Systems*, Aug. 1992.

[8] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," *Proc. Seventh National Conf. on Artificial Intelligence (AAAI-88)*, 1988, pp. 49-54.

[9] K. Decker, V. Lesser, R. Whitehair, "Extending a Blackboard Architecture for Approximate Processing," *The Journal of Real-Time Systems*, **2**, 1989, pp. 47-49.

[10] P. Goli, J.F. Kurose, D. Towsley, "Approximate Minimum Laxity Scheduling Algorithms for Real-Time Systems", COINS Technical Report TR-90-88.

[11] J. Hong, X. Tan, D. Towsley, "The Binary Simulation of the Minimum Laxity and the Earliest Deadline Scheduling Policies for Real-Time Systems", COINS Technical Report 89-70.

[12] J. Hong, X. Tan, D. Towsley, "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System," *IEEE Transactions on Computers*, **C-38**, 12, 1736-1744, December 1989.

[13] R. H. Katz, G. Gibson, and D. A. Patterson, "Disk System Architectures for High Performance Computing," *Proc. of the IEEE,* Vol.77, No.12, pp.1842-1858, Dec. 1989.

[14] C. M. Krishna and A. D. Singh, "Reliability of Voted-Time-Redundancy Real-Time Systems," to appear in *IEEE Trans. Reliability.*

[15] C.M. Krishna, A.D. Singh, "Modeling Correlated Transient Failures in Fault-Tolerant Systems," *Proc. Int'l Symposium on Fault-Tolerant Computing*, 1989.

[16] C. M. Krishna, "The Impact of Workload on the Reliability of Real-Time Processor Triads," to appear in *Micro. Rel.*

[17] J.F. Kurose, "Performance Analysis of Minimum Laxity Scheduling in Discrete Time Queueing Systems," submitted to *Performance Evaluation.*

[18] Y.-H. Lee, C.M. Krishna, "Optimal Scheduling of Signature Analysis for VLSI Testing," to appear in *IEEE Trans. Computers.*

[19] J. Liu, in "Report on the Embedded AI Languages Workshop, Ann Arbor, MI 1988," R. Volz, T. Mudge, G. Lindstorm (ed.), University of Michigan, Jan 27, 1990.

[20] P.Nain, D. Towsley, "Properties of the $ML(n)$ policy for scheduling jobs with real-time constraints", *IEEE Trans. on Computers*, **41**, 10, pp. 1271-1278, Oct. 1992.

[21] R. Nelson, A.N. Tantawi, D. Towsley, "The Order Statistics of the Sojourn Time s of Customers that Form a Single Batch in the M^X/Mc Queue", COINS Technical Report 91-05.

[22] R. Nelson, D. Towsley, "A Performance Evaluation of Several Priority Policies for Parallel Processing Systems", to appear in *J. ACM.*

[23] A. D. Singh and C. M. Krishna, "On Optimizing Wafer-Probe Testing for Product Quality Using Die-Yield Prediction," (with A. D. Singh), to appear in *IEEE Trans. Computer-Aided Design.*

[24] D. Towsley, S. Chen, "Bounds on the Performance of Two Server Fork/Join Queueing Systems," submitted to *Performance Evaluation.*

[25] D. Towsley, J.F. Kurose, "Disk Array Architectures for Real-Time Systems", *Proc. Annual ONR workshop on Real-Time Systems*, Oct. 1992.

[26] D. Towsley, S. Panwar, "Optimality of the stochastic earliest deadline policy for the G/M/c Queue Serving Customers with Deadlines", COINS Technical Report 91-61.

[27] D. Towsley, S. Tripathi, "A Single Server Priority Queue with Server Failures and Queue Flushing", *Operations Research Letters*, July 1991.

[28] N. Vaidya, A.D. Singh, C.M. Krishna, "Allocating Debug Time to Fault-Tolerant Software Modules," forthcoming ECE Technical Report.

[29] W. Zhao, J. A. Stankovic, "Performance evaluation of FCFS and improved FCFS scheduling for dynamic real-time computer systems," *Proc. Real-Time Systems Symposium*, 156-165, Dec. 1989.

Publications - ONR Grant No. N00014-87-K-0796
Design and Analysis of Scheduling Policies for Real-Time Computer Systems
J.F. Kuorse, C.M. Krishna, D. Towsley

1 Journal Articles:

- J. Hong, X. Tan, D. Towsley. "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System", *IEEE Transactions on Computers*, **C-38**, 12, 1736-1744, December 1989.
- P. Heidelberger, D. Towsley. "Sensitivity Analysis from Sample Paths Using Likelihoods", *Management Science*, **35**, 1475-1488, December 1989.
- D. Towsley, G. Rommel, J.A. Stankovic. "Analysis of Fork-Join Program Response Times on Multiprocessors", *IEEE Transactions on Parallel and Distributed Processing*, **1**, 3, 286-303, July 1990.
- F. Baccelli, D. Towsley. "Sojourn Times Under Processor Sharing are Associated", *Queueing Systems and their Applications*, **7**, 269-282, 1990.
- C.M. Krishna, Y.-H. Lee. "A Study of Two-Phase Service", *Operations Research Letters*, **9**, pp. 91-97, 1990.
- D. Towsley, F. Baccelli. "Comparisons of Service Disciplines in a Tandem Queueing Network with Real-Time Constraints", *Operations Research Letters*, **10**, 49-55, February 1991.
- S.-Z. Chen, J.A. Stankovic, J.F. Kurose, D. Towsley. "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems", *Real Time Systems*, **3**, 307–336, September 1991.
- Y. Dallery, D. Towsley. "Symmetry Property of the Throughput in Closed Tandem Queueing Networks with Finite Buffers", *Operations Research Letters*, **10**, 9, 541-547, December 1991.
- D. Sitaram, I. Koren and C. M. Krishna. "A Random, Distributed Algorithm to Embed Trees in Partially Faulty Processor Arrays", *J. Parallel and Distributed Computing*, Vol. 12, pp. 1-11, 1991.
- C. M. Krishna and Y.-H. Lee. "Guest Editors' Introduction: Real-Time Systems", *IEEE Computer*, Vol. 24, No. 5, pp. 10–11, 1991.
- Y.-H. Lee, C.M. Krishna. "Optimal Scheduling of Signature Analysis for VLSI Testing", *IEEE Trans. Computers* **40**, 336–341, March 1991.
- D. Towsley, S. Tripathi. "A Single Server Queue with Server Failures and Queue Flushing", *Operations Research Letters*, **10**, 353-362, August, 1991.
- Y. Dallery, D. Towsley. "Symmetry Property of the Throughput in Closed Tandem Queueing Networks with Finite Buffers", *Operations Research Letters*, **10**, 9, 541-547, December 1991.

- P.Nain, D. Towsley. "Comparison of Hybrid Minimum Laxity/First-in-First-Out Scheduling policies for Real-Time Multiprocessors", *IEEE Trans. on Computers*, **41**, 10, 1271-1278, Oct. 1992.

2 Book Chapters:

- J.F. Kurose, D. Towsley, C.M. Krishna. "Design and Analysis of Processor Scheduling Policies for Real-Time Systems", chapter in Foundations of Real-Time Computing: Scheduling and Resource Allocation.
- K.G. Shin and C.M. Krishna. "New Performance Measures for Real-Time Digital Computer Controls and Their Applications", in Advances in Control and Dynamic Systems, Academic Press.
- J.A. Stankovic, K. Ramamritham, D. Towsley. "Scheduling in Real-Time Transactiion Systems", chapter in Foundations of Real-Time Computing: Scheduling and Resource Allocation.

3 Articles in Proceedings:

- S. Chen, D. Towsley. "Performance of a Mirrored Disk in a Real-Time Transaction System", *Proc. 1991 ACM SIGMETRICS Conference*, May 1991.
- Q. Yu, D. Towsley, P. Heidelberger. "Time Driven Parallel Simulation of Multistage Interconnection Networks", *Proc. Distr. Simulation part of Eastern Multiconf.*, Tampa, FL, March 1989, pp. 191-196.
- C.M. Krishna, A.D. Singh. "Modeling Correlated Transient Failures in Fault-Tolerant Systems", *Proc. Int'l Symposium on Fault-Tolerant Computing*, 1989.
- P. Goli, P. Heidelberger, D. Towsley, Q. Yu. "Processor Allocation Schemes for Time Driven Parallel Simulation of Multistage Interconnection Networks", *Proceedings of the Distributed Simulation part of the Eastern Multiconference*, 1990.
- D. Towsley, S.S. Panwar. "On the Optimality of Minimum Laxity and Earliest Deadline Scheduling for Real-Time Multiprocessors", *Proceedings of the Euromicro'90 Workshop on Real Time*, pp. 17-24, June 1990.
- D. Towsley, S.-Z. Chen, S.P. Yu. "Performance Analysis of A Fault Tolerant Mirrored Disk System", *Proceedings of PERFORMANCE'90*, September 1990.
- P.Nain, D. Towsley. "Properties of the $ML(n)$ Policy for Scheduling Jobs with Real-Time Constraints", *Proceedings of 29-th IEEE Conference on Control and Decision*,Vol. 2, pp. 915-920, Honolulu, Hawaii, December 1990.

4 Journal Articles (to appear):

- S.-Z. Chen and D. Towsley. "Raid 5 vs. Parity Stripping: Their Design and Evaluation", to appear in *Journal of Parallel and Distributed Computing*, January 1993.
- F. Baccelli, Z. Liu, D. Towsley. "Optimal Scheduling of Parallel Processing Systems with Real-Time Constraints", to appear in *J. ACM.*
- R. Nelson, D. Towsley. "A Performance Evaluation of Several Priority Policies for Parallel Processing Systems", to appear in *J. ACM.*
- C. M. Krishna and A. D. Singh. "Reliability of Voted-Time-Redundancy Real-Time Systems", to appear in *IEEE Trans. Reliability.*
- C. M. Krishna. "The Impact of Workload on the Reliability of Real-Time Processor Triads", to appear in *Micro. Rel.*
- J.F. Kurose. "Performance Analysis of Minimum Laxity Scheduling in Discrete Time Queueing Systems", to appear in *Performance Evaluation.*
- A. D. Singh and C. M. Krishna. "On Optimizing Wafer-Probe Testing for Product Quality Using Die-Yield Prediction", to appear in *IEEE Trans. Computer-Aided Design.*
- D. Towsley. "A Study of a Queueing System with Three-Phase Service", to appear in *Operations Research Letters.*
- Y. Dallery, Z. Liu, D. Towsley. "Equivalence, Reversibility, Symmetry and Concavity Properties in Fork/Join Queueing Networks with Blocking", to appear in *Journal of the ACM.*

5 Book Chapters (to appear):

- Y.-H. Lee and C. M. Krishna, editors. *Readings in Real-Time Systems,* IEEE Computer Society Press, to appear.
- C. M. Krishna. "Architectural Aspects of Real-Time Systems", Encyclopedia of Computer Science and Technology, to appear.

6 Articles in Conference Proceedings (to appear):

- J. Dey, J.F. Kurose, D. Towsley, C.M. Krishna, M. Girkar. "Efficient On-line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks", to appear in *Proc. of 1993 ACM SIGMETRICS Conf.*

7 Technical Reports:

- S. Chen, J.A. Stankovic, J. Kurose, D. Towsley. "Perf. Eval. of Two New Disk Scheduling Algorithms for Real-Time Systems", COINS TR90-77.
- S. Chen, D. Towsley. "Perf. of a Mirrored Disk in a Real-Time Transaction System", COINS TR91-07.
- S. Chen, D. Towsley. "A Queueing Analysis of RAID Architectures", COINS TR91-71.
- S. Chen, D. Towsley. "RAID 5 vs. parity Striping: Their Design and Evaluation", COINS TR92-13.
- S.Chen, D. Towsley. "Scheduling Customers in a Non-Removal Real-Time System with an Application to Disk Scheduling", COINS TR92-58.
- S. Chen, D. Towsley. "A Perf. Eval. of RAID Architectures", (Replaces TR91-71), COINS TR92-67.
- Y. Dallery, Z. Liu, D. Towsley. "Equivalence, Reversibility and Symmetry Properties in Fork/Join Queueing Networks with Blocking", COINS TR90-78.
- Y. Dallery, Z. Liu, D. Towsley. "Properties of Fork/Join Networks with Blocking under Various Operating Mechanisms", COINS TR92-39.
- Y. Dallery, D. Towsley. "Symmetry Property of the Throughput in Closed Tandem Queueing Networks with Finite Buffers", COINS TR90-40.
- P. Goli, J. Kurose, D. Towsley. "Approximate Minimum Laxity Scheduling Algorithms for Real-Time Systems", COINS TR90-88.
- J. Hong, X. Tan, D.Towsley. "The Binary Simulation of the Minimum Laxity and Earliest Deadline Scheduling Policies for Real Time Systems", COINS TR89-70.
- J. Hong, X. Tan, D. Towsley. "A Performance Analysis of Minimum Laxity and Earliest Deadline Scheduling in a Real-Time System", COINS TR89-71.
- R. Nelson, A. Tantawi, D. Towsley. "The Order Statistics of the Sojourn Times of Customers that Form a Single Batch in the Mx/M/c Queue", COINS TR 91-05.
- R. Nelson, D. Towsley. "A performance Evaluation of Several Priority Policies for Parallel Processing Systems", COINS TR91-32.
- D. Towsley, S. Chen, S.P. Yu. "Performance Analysis of a Fault Tolerant Mirrored Disk System", COINS TR91-04.
- D. Towsley, S. Panwar. "Comparison of Service and Buffer Overflow Policies for Mult. Server Queues that Serve Customers with Deadlines", COINS TR89-72.
- D. Towsley, S. Panwar. "Optimality of the Stochastic Earliest Deadline Policy for the G/M/c Queue Serving Customers with Deadlines", COINS TR91-61.